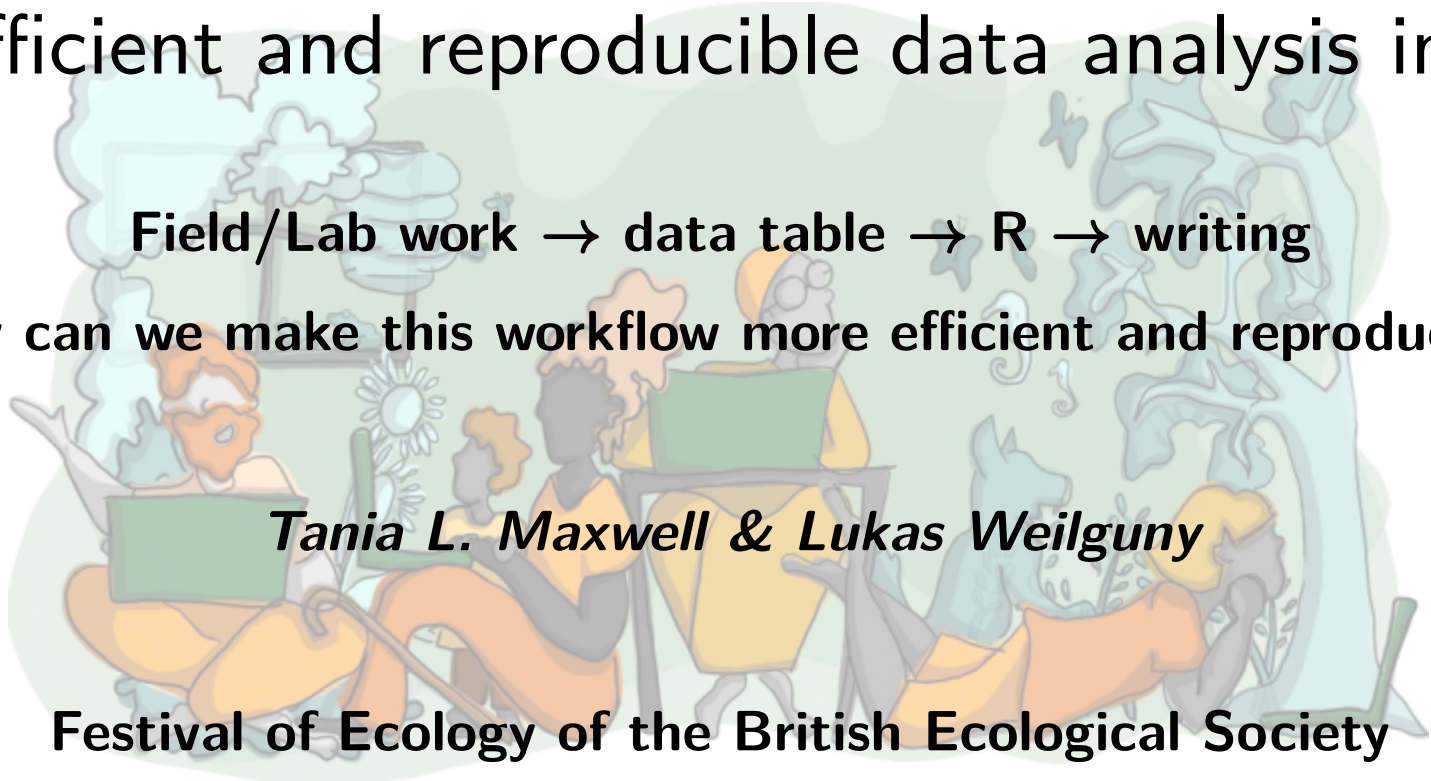
Efficient and reproducible data analysis in R



Field/Lab work → data table → R → writing

How can we make this workflow more efficient and reproducible?

Tania L. Maxwell & Lukas Weilguny

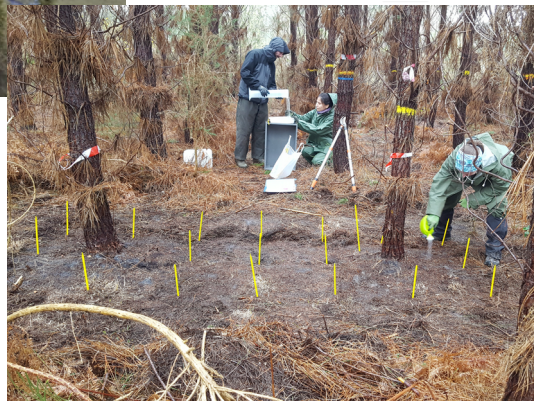
Festival of Ecology of the British Ecological Society

December 15th, 2020 | 14:00 – 15:30 GMT

Tania L. Maxwell

université de BORDEAUX

UNIVERSITÉ LAVAL



INRAE

Lukas Weilguny

EMBL-EBI

UNIVERSITY OF CAMBRIDGE

Outline

14:00 – Introduction from chairs

14:10 – Diving into 'tidy data' and dplyr and custom functions

14:35 – Reproducibility: good practices and generating reports with R Markdown

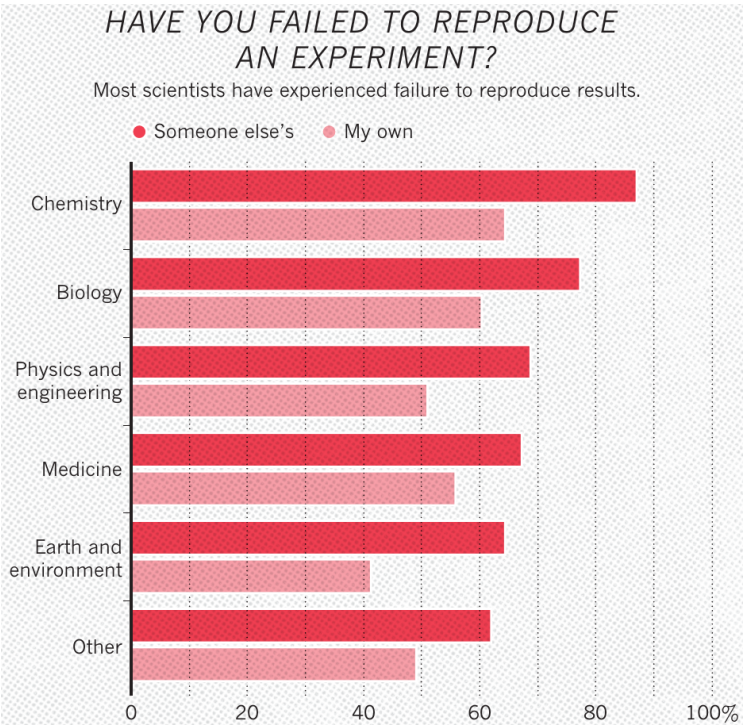
15:00 – Break

15:05 – Breakout sessions - teamwork

15:20 – Feedback from breakout rooms, summary and conclusions

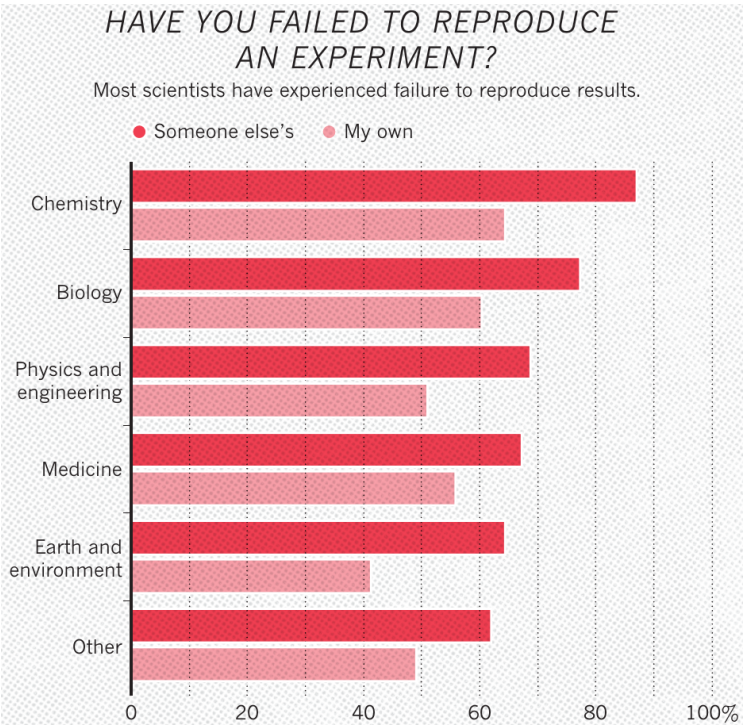
Have you ever tried and failed to reproduce your own or someone else's work?

Have you ever tried and failed to reproduce your own or someone else's work?



Baker. Nature 533, 2016

Have you ever tried and failed to reproduce your own or someone else's work?



Baker. Nature 533, 2016

Open & reproducible science:

- Open data
- **Open tools & code**
- Open access

Diving into ‘tidy data’ and dplyr and custom functions

Why should we care about reproducibility?

- Reproducibility study in psychology: only 39 of 100 studies could be reproduced. (Baker, Nature 2015.)
- Estimated $28 billion spent each year on biomedical research in U.S that cannot be repeated successfully (Freedman et al. PloS Biol. 2015)

Why should we care about reproducibility?

- Reproducibility study in psychology: only 39 of 100 studies could be reproduced. (Baker, Nature 2015.)
- Estimated $28 billion spent each year on biomedical research in U.S that cannot be repeated successfully (Freedman et al. PloS Biol. 2015)

What is needed for robust and reproducible research

- Methods, code, data must be available and well documented
- Cautious attitude towards data and code
- Taking time to develop fequently used scripts into functions or tools
- Let lab members or collaborators use your code!

Why should we care about reproducibility?

- Reproducibility study in psychology: only 39 of 100 studies could be reproduced. (Baker, Nature 2015.)
- Estimated $28 billion spent each year on biomedical research in U.S that cannot be repeated successfully (Freedman et al. PloS Biol. 2015)

What is needed for robust and reproducible research

- Methods, code, data must be available and well documented
- Cautious attitude towards data and code
- Taking time to develop fequently used scripts into functions or tools
- Let lab members or collaborators use your code!

What's the benefit?

- Easily re-run analysis
- Revisiting projects in the future
- Modularize and re-use
- Easier collaborations

Open (data) is not enough

based on Chen at al. Nature Physics 15, 2019

Terms	**Purpose**	**Example**
Rerun & Repeat	Develop and review	Intergral to the analysis workflow; ease of updating with new measurements
Replicate & Reproduce	Certify & Compare	Possibility for different researchers to arrive at the same results
Reuse	Transfer & new purpose	Reusing code for independent study or more recent dataset

$\rightarrow$ Reusability & reproducibility requires code, workflows and comments!

Good practice for writing code I

- Write code for humans, write data for computers
  - Annotate purpose & authors
- Explicit dependencies
- Avoid 'hard-coding' variables
  - Define paths at beginning
  - Use relative paths
- Clear, concise comments
  - What & why, not how

```
1  # This is to replicate the analysis of section 3 of
2  # our 2019 MEE paper. Developed by T. Maxwell, L. Weilguny
3
4  library(ggplot2)
5  library(vegan)
6  library(alphahull)  # version 2.2
7
8  input_file <- "data/data.csv"
9  output_file <- "data/results.csv"
10
11 # read input
12 input_data <- read.csv(input_file)
13 # number of samples
14 samples <- nrow(input_data)
15 .
16 .
17 # run the last step of analysis
18 results <- some_function(input_file, sample_number)
19 # write results to output file
20 write.table(results, output_file)
```

Good practice for writing code I

- Write code for humans, write data for computers
  - Annotate purpose & authors
- Explicit dependencies
- Avoid 'hard-coding' variables
  - Define paths at beginning
  - Use relative paths
- Clear, concise comments
  - What & why, not how

```
1  # This is to replicate the analysis of section 3 of
2  # our 2019 MEE paper. Developed by T. Maxwell, L. Weilguny
3
4  library(ggplot2)
5  library(vegan)
6  library(alphahull)  # version 2.2
7
8  input_file <- "data/data.csv"
9  output_file <- "data/results.csv"
10
11 # read input
12 input_data <- read.csv(input_file)
13 # number of samples
14 samples <- nrow(input_data)
15 .
16 .
17 # run the last step of analysis
18 results <- some_function(input_file, sample_number)
19 # write results to output file
20 write.table(results, output_file)
```

Good practice for writing code I

- Write code for humans, write data for computers
  - Annotate purpose & authors
- Explicit dependencies
- Avoid 'hard-coding' variables
  - Define paths at beginning
  - Use relative paths
- Clear, concise comments
  - What & why, not how

```
1  # This is to replicate the analysis of section 3 of
2  # our 2019 MEE paper. Developed by T. Maxwell, L. Weilguny
3
4  library(ggplot2)
5  library(vegan)
6  library(alphahull)  # version 2.2
7
8  input_file <- "data/data.csv"
9  output_file <- "data/results.csv"
10
11 # read input
12 input_data <- read.csv(input_file)
13 # number of samples
14 samples <- nrow(input_data)
15 .
16 .
17 # run the last step of analysis
18 results <- some_function(input_file, sample_number)
19 # write results to output file
20 write.table(results, output_file)
```

Good practice for writing code I

- Write code for humans, write data for computers
  - Annotate purpose & authors
- Explicit dependencies
- Avoid 'hard-coding' variables
  - Define paths at beginning
  - Use relative paths
- Clear, concise comments
  - What & why, not how

```
# This is to replicate the analysis of section 3 of
# our 2019 MEE paper. Developed by T. Maxwell, L. Weilguny

library(ggplot2)
library(vegan)
library(alphahull)  # version 2.2

input_file <- "data/data.csv"
output_file <- "data/results.csv"

# read input
input_data <- read.csv(input_file)
# number of samples
samples <- nrow(input_data)
.
.
# run the last step of analysis
results <- some_function(input_file, sample_number)
# write results to output file
write.table(results, output_file)
```

Good practice for writing code II

- Choose a style and be consistent
  (e.g. style.tidyverse.org)
  - Naming things (variables, functions)
  - Organization
  - Appearance (indentation, spacing, brackets)
  - Comments
  - Line lengths (<80)

- Modularize into smallest units
  - One specific job per function
  - Keep them in separate file & re-use
  - use functions within functions instead of long, nested structures

Good practice for writing code II

- Choose a style and be consistent (e.g. style.tidyverse.org)
  - Naming things (variables, functions)
  - Organization
  - Appearance (indentation, spacing, brackets)
  - Comments
  - Line lengths (<80)

- Modularize into smallest units
  - One specific job per function
  - Keep them in separate file & re-use
  - use functions within functions instead of long, nested structures

Bad example

Code example based on nicercode.github.io/guides/repeating-things/

```
for(n in 1:n.spp)
{
  Ind = unique(Raw[Raw$SPP==as.character(sp.list$SPP[n]), "INDIV"]);
  I =    length(Ind);
  par(mfrow=c(I,1), oma=c(5,5,5, 2), mar=c(3, 0, 0, 0));
  for(i in 1:I){
    Dat = subset(Raw, Raw$SPP==as.character(sp.list$SPP[n]) & Raw$INDIV==Ind[i])
    Y_ax =seq(0, 35, 10); Y_ax2=seq(0, 35, 5);
    X_ax =seq(0, max(Dat$LL), 0.2); X_ax2 =seq(0, max(Dat$LL), 0.1);
    plot(1:2, 1:2, type="n",log="", axes=F,ann=F, xlim = c(-0.05, max(Dat$LL)+0.05), xaxs="i")
    X<-Dat$AGE;  Xout<-data.frame(X = c(0,Dat$LL[1]))
    Y<-Dat$S2AV_L;
    points(X,Y,type="p", pch=Fig$Symb[2], cex=1.3, col= Fig$Cols[2]);
    R<-lm( Y~ X); points(Xout$X, predict(R, Xout), type="l", col= Fig$Cols[2], lty = "dotted")
  }
  mtext(expression(paste("Intercepted light (mol ", m^{-2},d^{-1},")")), side = 2, line = 3)
  mtext(expression(paste("Leaf age (yrs)")), side = 1, line = 0.2, outer = T, adj = 0.5, cex =1.2)
}
rm(R, Ind, I, i, X, X_ax, X_ax2, Y_ax, Y_ax2, Y, Xout, Dat)

```

Good practice for writing code II

- Choose a style and be consistent (e.g. style.tidyverse.org)
  - Naming things (variables, functions)
  - Organization
  - Appearance (indentation, spacing, brackets)
  - Comments
  - Line lengths (<80)

- Modularize into smallest units
  - One specific job per function
  - Keep them in separate file & re-use
  - use functions within functions instead of long, nested structures

Bad example

Code example based on nicercode.github.io/guides/repeating-things/

```
for(n in 1:n.spp)
{
  Ind = unique(Raw[Raw$SPP==as.character(sp.list$SPP[n]), "INDIV"]);
  I =    length(Ind);
  par(mfrow=c(I,1), oma=c(5,5,5, 2), mar=c(3, 0, 0, 0));
  for(i in 1:I){
    Dat = subset(Raw, Raw$SPP==as.character(sp.list$SPP[n]) & Raw$INDIV==Ind[i])
    Y_ax =seq(0, 35, 10); Y_ax2=seq(0, 35, 5);
    X_ax =seq(0, max(Dat$LL), 0.2); X_ax2 =seq(0, max(Dat$LL), 0.1);
    plot(1:2, 1:2, type="n",log="", axes=F,ann=F, xlim = c(-0.05, max(Dat$LL)+0.05), xaxs="i")
    X<-Dat$AGE;  Xout<-data.frame(X = c(0,Dat$LL[1]))
    Y<-Dat$S2AV_L;
    points(X,Y,type="p", pch=Fig$Symb[2], cex=1.3, col= Fig$Cols[2]);
    R<-lm( Y~ X); points(Xout$X, predict(R, Xout), type="l", col= Fig$Cols[2], lty = "dotted")
  }
  mtext(expression(paste("Intercepted light (mol ", m^{-2},d^{-1},")")), side = 2, line = 3)
  mtext(expression(paste("Leaf age (yrs)")), side = 1, line = 0.2, outer = T, adj = 0.5, cex =1.2)
}
rm(R, Ind, I, i, X, X_ax, X_ax2, Y_ax, Y_ax2, Y, Xout, Dat)

#####################  better

makePlot <- function(species, data){
  ...  # well commented code to make a plot
}

# loop over unique entries of the column species
for (i in unique(raw_measurements$Species))
  makePlot(i, data = raw_measurements)

```

Good practice for writing code II

- Choose a style and be consistent (e.g. style.tidyverse.org)
  - Naming things (variables, functions)
  - Organization
  - Appearance (indentation, spacing, brackets)
  - Co
  - Lin

> "being *readable*, *reusable*, and *testable* are all side effects of writing *modular* code"
> Wilson et al. PloS Comp Biol. 2017

- Modularize into smallest units
  - One specific job per function
  - Keep them in separate file & re-use
  - use functions within functions instead of long, nested structures

Code example based on nicercode.github.io/guides/repeating-things/

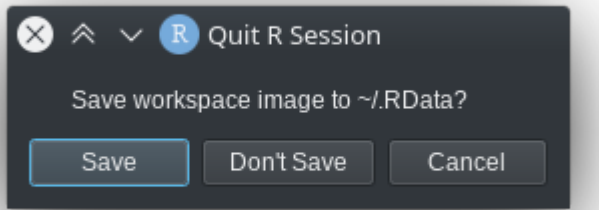
```
2  for(n in 1:n.spp)
3  {
4    Ind = unique(Raw[Raw$SPP==as.character(sp.list$SPP[n]), "INDIV"]);
5    I =    length(Ind);
6    par(mfrow=c(I,1), oma=c(5,5,5, 2), mar=c(3, 0, 0, 0));
7    for(i in 1:I){
8      Dat = subset(Raw, Raw$SPP==as.character(sp.list$SPP[n]) & Raw$INDIV==Ind[i])
9      Y_ax =seq(0, 35, 10); Y_ax2=seq(0, 35, 5);
10     X_ax =seq(0, max(Dat$LL), 0.2); X_ax2 =seq(0, max(Dat$LL), 0.1);
11     plot(1:2, 1:2, type="n",log="", axes=F,ann=F, xlim = c(-0.05, max(Dat$LL)+0.05), xaxs="i")
12     X<-Dat$AGE;  Xout<-data.frame(X = c(0,Dat$LL[1]))
                                                                    ;
                                                    g$Cols[2], lty = "dotted")
                                                                    ")), side = 2, line = 3)
18   mtext(expression(paste("Leaf age (yrs)")), side = 1, line = 0.2, outer = T, adj = 0.5, cex =1.2)
19 }
20 rm(R, Ind, I, i, X, X_ax, X_ax2, Y_ax, Y_ax2, Y, Xout, Dat)
21
22 ######################
23
24 makePlot <- function(species, data){
25   ...  # well commented code to make a plot
26 }
27
28 # loop over unique entries of the column species
29 for (i in unique(raw_measurements$Species))
30   makePlot(i, data = raw_measurements)
31
```

Good practice for writing code III

- Don't save session history (RData)
  - start in clean environment each time

- for anything that involves random numbers use `set.seed()`

- keep record of `sessionInfo()`, e.g. at end of script

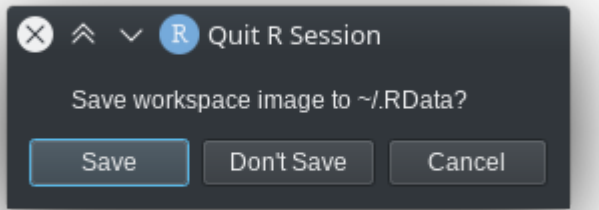


Good practice for writing code III

- Don't save session history (RData)
  - start in clean environment each time

- for anything that involves random numbers use `set.seed()`

- keep record of `sessionInfo()`, e.g. at end of script



Good practice for writing code III

- Don't save session history (RData)
  - start in clean environment each time
- for anything that involves random numbers use `set.seed()`
- keep record of `sessionInfo()`, e.g. at end of script

```
> sessionInfo()
R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: KDE neon User Edition 5.20

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0

locale:
 [1] LC_CTYPE=en_GB.UTF-8    LC_NUMERIC=C           LC_TIME=en_GB
 [6] LC_MESSAGES=en_GB.UTF-8 LC_PAPER=en_GB.UTF-8   LC_NAME=C
[11] LC_MEASUREMENT=C        LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   ba

other attached packages:
[1] alphahull_2.2   vegan_2.5-7     lattice_0.20-41 permute_0.9-5

loaded via a namespace (and not attached):
 [1] tidyselect_1.1.0      xfun_0.19             purrr_0.3.4
 [7] generics_0.0.2        spatstat.utils_1.17-0 htmltools_0.5.0
[13] R.oo_1.23.0           spatstat.data_1.4-3   pillar_1.4.4
[19] R.utils_2.9.2         sp_1.4-2              sgeostat_1.0-27
```

Good practice for writing code III

- Don't save session history (RData)
  - start in clean environment each time
- for any
  use `set.seed()`
- keep record of `sessionInfo()`,
  e.g. at end of script

Do you have any other suggestions that are important to you?

```
> sessionInfo()
R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: KDE neon User Edition 5.20

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0

locale:
 [1] LC_CTYPE=en_GB.UTF-8    LC_NUMERIC=C                LC_TIME=en_GB
                                                 TF-8    LC_NAME=C
                                                 N=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   ba

other attached packages:
[1] alphahull_2.2   vegan_2.5-7     lattice_0.20-41 permute_0.9-5

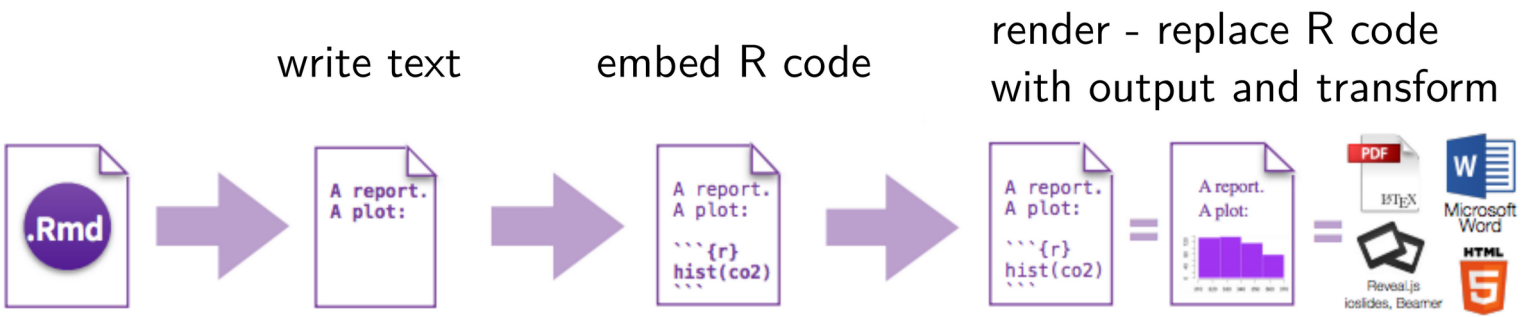
loaded via a namespace (and not attached):
 [1] tidyselect_1.1.0      xfun_0.19             purrr_0.3.4
 [7] generics_0.0.2        spatstat.utils_1.17-0 htmltools_0.5.0
[13] R.oo_1.23.0           spatstat.data_1.4-3   pillar_1.4.4
[19] R.utils_2.9.2         sp_1.4-2              sgeostat_1.0-27
```

Literate programming can help us make research reproducible

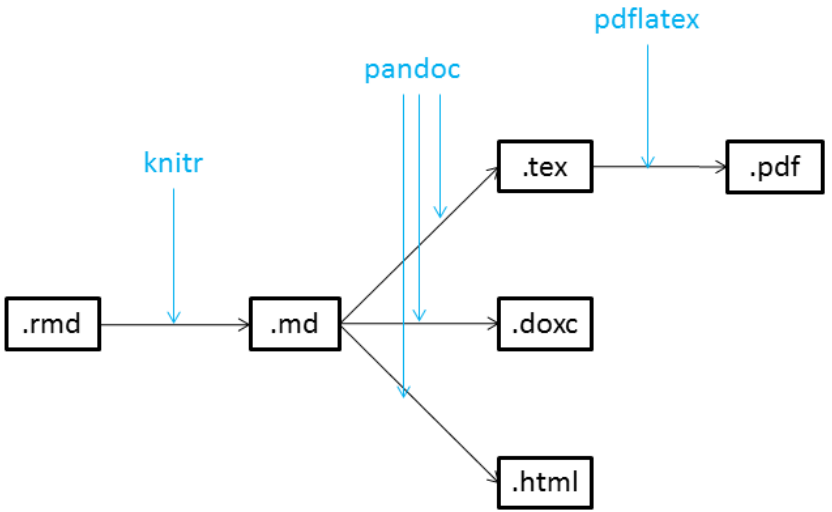
- Integrating text, figures & code
- Upon finding errors or adding new data:
  - simply re-compile the report
  - No need to reconstruct figures, paste into Word document
- Continuous editing & updating
- Can be used with versioning tools like git

Literate programming can help us make research reproducible

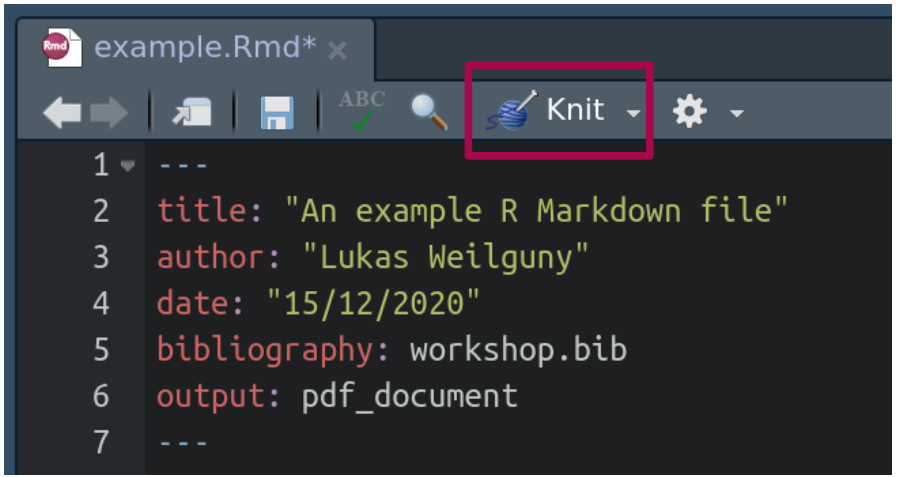
- Integrating text, figures & code
- Upon finding errors or adding new data:
  - simply re-compile the report
  - No need to reconstruct figures, paste into Word document
- Continuous editing & updating
- Can be used with versioning tools like git



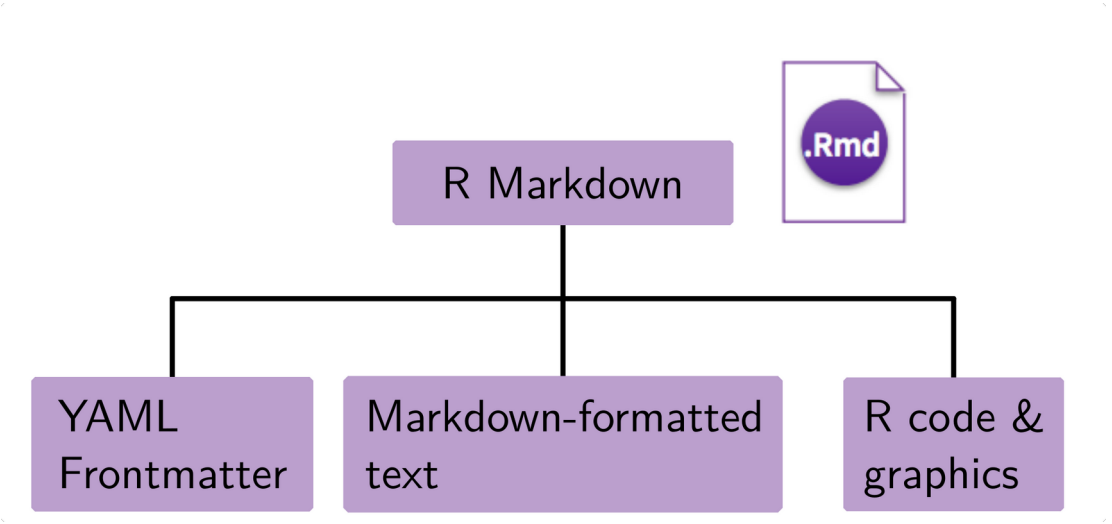
A simple as clicking one button



A simple as clicking one button



Structure of an R Markdown document



Structure of an R Markdown document

YAML header
(metadata)

Markdown text
(static content)

R code
(dynamic content and
figure generation)

````
---
title: "An example R Markdown file"
author: "Lukas Weilguny"
date: "15/12/2020"
output: pdf_document
---

# Section 1

Here we write with *Markdown syntax* [@Chen2019].

```{r}
today <- Sys.Date()
paste('Thank you for participating on ', today)
```

# Section 2

R code can also be used inline. E.g. variables assigned in code chunks: `r today`

````

Structure of an R Markdown document

YAML header
(metadata)

Markdown text
(static content)

R code
(dynamic content and
figure generation)

```
---
title: "An example R Markdown file"
author: "Lukas Weilguny"
date: "15/12/2020"
output: pdf_document
---

# Section 1

Here we write with *Markdown syntax* [@Chen2019].

```{r}
today <- Sys.Date()
paste('Thank you for participating on ', today)
```

Section 2

R code can also be used inline. E.g. variables assigned in code chunks: `r today`

```

# Structure of an R Markdown document

YAML header
(metadata)

Markdown text
(static content)

R code
(dynamic content and
figure generation)

```
---
title: "An example R Markdown file"
author: "Lukas Weilguny"
date: "15/12/2020"
output: pdf_document

Section 1

Here we write with *Markdown syntax* [@Chen2019].

```{r}
today <- Sys.Date()
paste('Thank you for participating on ', today)
```

Section 2

R code can also be used inline. E.g. variables assigned in code chunks: `r today`
```

Structure of an R Markdown document

YAML header
(metadata)

Markdown text
(static content)

R code
(dynamic content and
figure generation)

```
---
title: "An example R Markdown file"
author: "Lukas Weilguny"
date: "15/12/2020"
output: pdf_document
---

# Section 1

Here we write with *Markdown syntax* [@Chen2019].

```{r}
today <- Sys.Date()
paste('Thank you for participating on ', today)
```

Section 2

R code can also be used inline. E.g. variables assigned in code chunks: `r today`

```

# Structure of an R Markdown document

YAML header
(metadata)

Markdown text
(static content)

R code
(dynamic content and
figure generation)


```
1 ---
2 title: "An example R Markdow
3 author: "Lukas Weilguny"
4 date: "15/12/2020"
5 output: pdf_document
6 ---
7
8 # Section 1
9
10 Here we write with *Markdown syntax* [@Chen2019].
11
12 ```{r}
13 today <- Sys.Date()
14 paste('Thank you for participating on ', today)
15 ```
16
17 # Section 2
18
19 R code can also be used inline. E.g. variables assigned in code chunks: `r today`
20
```

An example R Markdown file

Lukas Weilguny

15/12/2020

Section 1

Here we write with *Markdown syntax* [@Chen2019].

```
today <- Sys.Date()
paste('Thank you for participating on ', today)
```

```
## [1] "Thank you for participating on  2020-12-07"
```

Section 2

R code can also be used inline. E.g. variables assigned in code chunks: 2020-12-07

How to write text in Markdown

```
# Markdown syntax

* normal text
* __bold text__
* _italicised text_
```

Markdown syntax

- normal text
- **bold text**
- *italicised text*

How to write text in Markdown

```
# Markdown syntax

* normal text
* __bold text__
* _italicised text_
```

Markdown syntax

- normal text
- **bold text**
- *italicised text*

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com

Learn more about Interactive Docs at shiny.rstudio.com/articles

LaTeX can be mixed with Markdown

```
# Typesetting equations

For example, the entropy $H(x) = -\sum p(x)\log p(x)$

# Managing bibliographies and citations

Sharing data is not sufficient! It is important to also document the analysis workflows [@Chen2019].
@Chen2019 advises how to ensure usability and longevity of research outcomes.

## References

```

LaTeX can be mixed with Markdown

```
# Typesetting equations

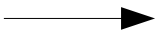
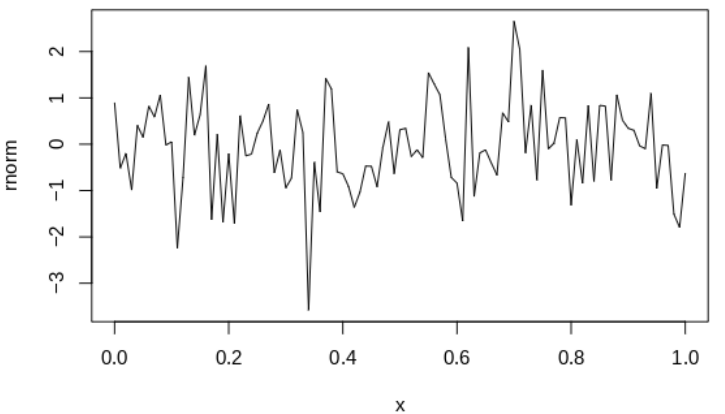
For example, the entropy $H(x) = -\sum p(x)\log p(x)$

# Managing bibliographies and citations

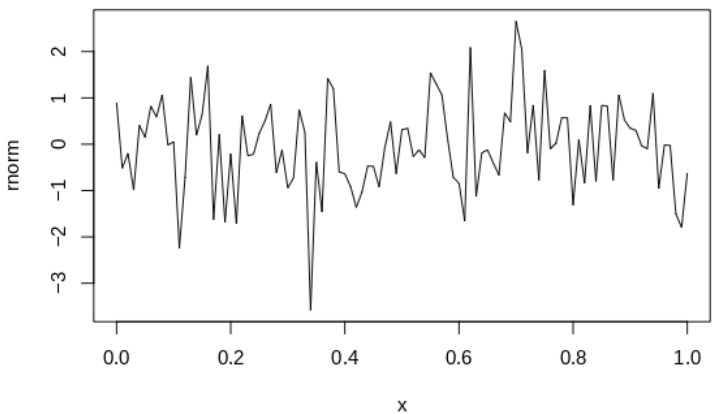
Sharing data is not sufficient! It is important to also document the analysis workflows [@Chen2019].
@Chen2019 advises how to ensure usability and longevity of research outcomes.

## References
```

Typesetting equations

For example, the entropy $H(x) = -\sum p(x) \log p(x)$

Managing bibliographies and citations

Sharing data is not sufficient! It is important to also document the analysis workflows (Chen et al. 2019). Chen et al. (2019) advises how to ensure usability and longevity of research outcomes.

References

Chen, Xiaoli, Sünje Dallmeier-Tiessen, Robin Dasler, Sebastian Feger, Pamfilos Fokianos, Jose Benito Gonzalez, Harri Hirvonsalo, et al. 2019. "Open Is Not Enough." *Nature Physics* 15 (2): 113–19. https://doi.org/10.1038/s41567-018-0342-2.

Managing bibliographies and citations

1. Common `.biblatex` format
   (automatically exported from your
   favourite reference manager)

```
@article{Chen2019,
    title   =   {Open is not enough},
    volume  =   {15},
    doi     =   {10.1038/s41567-018-0342-2},
    pages   =   {113--119},
    number  =   {2},
    year    =   {2019},
    journal =   {Nature Physics},
    author  =   {Chen, Xiaoli and Dallmeier-Tiessen, Sünje and Dasler,
    Robin and Feger, Sebastian and Fokianos, Pamfilos and Gonzalez, Jose
    Benito and Hirvonsalo, Harri and Kousidis, Dinos and Lavasa, Artemis
    and Mele, Salvatore and Rodriguez, Diego Rodriguez and Šimko, Tibor
    and Smith, Tim and Trisovic, Ana and Trzcinska, Anna and Tsanaktsidis,
    Ioannis and Zimmermann, Markus and Cranmer, Kyle and Heinrich, Lukas
    and Watts, Gordon and Hildreth, Michael and Lloret Iglesias, Lara and
    Lassila-Perini, Kati and Neubert, Sebastian}
}
```

2. Indicate bibliography in YAML header

```
1  ---
2  title: "An example R Markdown file"
3  author: "Lukas Weilguny"
4  date: "15/12/2020"
5  bibliography: workshop.bib
6  output: pdf_document
7  ---
```

3. Simple citation using `[@codename]` or `@codename` or use plug-in `citr` for graphical interface

Dynamic content: R Code chunks

````
An example how code and output

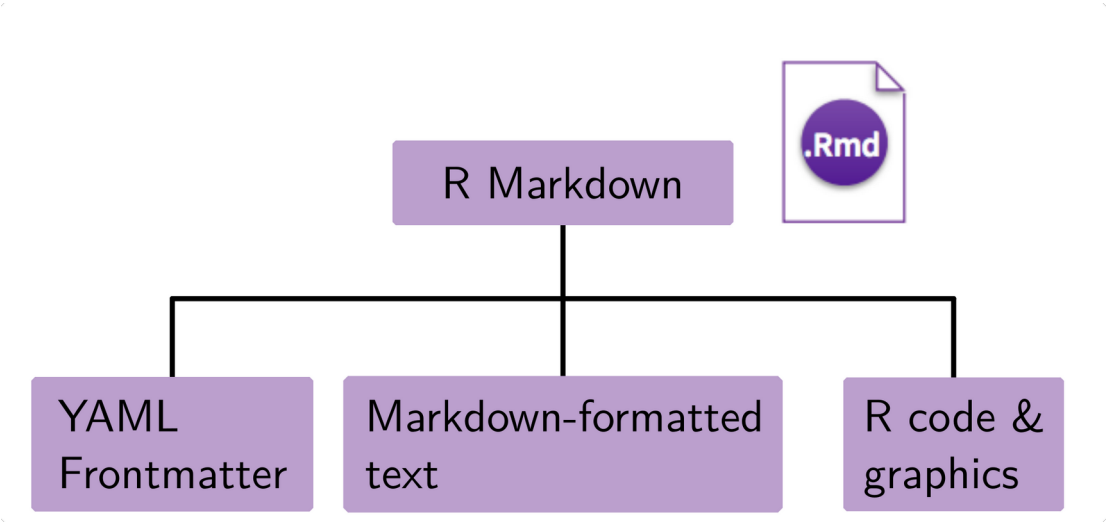
```{r, eval=TRUE, echo=TRUE, out.width="50%"}
random_numbers <- rnorm(100)
plot(rnorm)
```

are embedded in the rendered document
````



An example how code and output

```
random_numbers <- rnorm(100)
plot(rnorm)
```



are embedded in the rendered document

Dynamic content: R Code chunks

````
An example how code and output

```{r, eval=TRUE, echo=TRUE, out.width="50%"}
random_numbers <- rnorm(100)
plot(rnorm)
```

are embedded in the rendered document
````

Chunk options modify behaviour

An example how code and output

```r
random_numbers <- rnorm(100)
plot(rnorm)
```



are embedded in the rendered document

Chunk options control appearance of code and output

Chunk option	effect
`echo=FALSE`	do not show the code itself
`results="hide"`	do not print any results
`eval=FALSE`	print code but do not run
`warning=FALSE` `message=FALSE`	hide warnings
`fig.height=` `fig.width=`	control figure output
...	

Chunk options control appearance of code and output

Chunk option	effect
`echo=FALSE`	do not show the code itself
`results="hide"`	do not print any results
`eval=FALSE`	print code but do not run
`warning=FALSE` `message=FALSE`	hide warnings
`fig.height=` `fig.width=`	control figure output
...	

setting global chunk options:

````
```{r global_options, echo=FALSE}
knitr::opts_chunk$set(fig.path="Figs/", message=FALSE, warning=FALSE,
                      echo=FALSE, results="hide", fig.width=11)
```
````

Structure of an R Markdown document

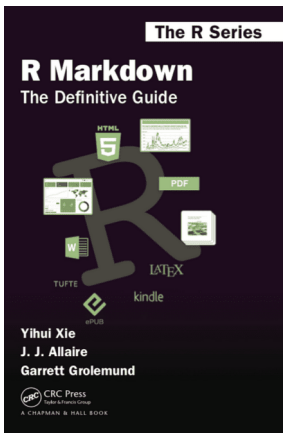


Further resources

- Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, et al. (2017)
  Good enough practices in scientific computing.
  PLOS Computational Biology 13(6). doi.org/10.1371/journal.pcbi.1005510

- Inspiration: https://rmarkdown.rstudio.com/gallery
  - Reports, presentations, websites, templates, books ...

- The Definitive Guide by Yihui Xie, J. J. Allaire, Garrett Grolemund
  - https://bookdown.org/yihui/rmarkdown/



Short break.. see you in 5 minutes

Breakout challenge

- Breakout rooms for teamwork
- a unique link for each room to work together on a piece of code
- The code will be poorly written and not re-usable
- Your challenge is to identify problems and make it re-usable
  (hint: look at variables, create functions, etc)

codeshare

Enables writing code simultaneously but not running
If you want to one person in the group could use Rstudio and share their screen

```r
nutrients <- data.frame(
  site=c("a","a","a","a","a","b","b","b","b","b"),
  hi_density=c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, TRUE),
  conc_N=c(0.26, 0.27, 0.18, 0.23, 1.47, 0.33, 0.52, 0.86, 0.53, 0.15),
  conc_P=c(0.14, 0.77, 0.34, 1.63, 0.59, 0.46, 0.90, 2.50, 0.97, 0.03))

# this is how our nutrients data frame would look like:
#    site   hi_density conc_N conc_P  ... ...
# 1     a         TRUE   0.26   0.14
# 2     a         TRUE   0.27   0.77
# 3     a        FALSE   0.18   0.34
# 4     a        FALSE   0.23   1.63
# 5     a        FALSE   1.47   0.59
# 6     b        FALSE   0.33   0.46
# 7     b         TRUE   0.52   0.90
# 8     b         TRUE   0.86   2.50
# 9     b        FALSE   0.53   0.97
# 10    b         TRUE   0.15   0.03
# ...
# ...


# Objective: find a modularized way to calculate the standard deviation
#            for all columns that contain concentration measurements
```

```r
nutrients <- data.frame(
  site=c("a","a","a","a","a","b","b","b","b","b"),
  hi_density=c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, TRUE),
  conc_N=c(0.26, 0.27, 0.18, 0.23, 1.47, 0.33, 0.52, 0.86, 0.53, 0.15),
  conc_P=c(0.14, 0.77, 0.34, 1.63, 0.59, 0.46, 0.90, 2.50, 0.97, 0.03))

# this is how our nutrients data frame would look like:
#    site   hi_density conc_N conc_P  ... ...
# 1     a         TRUE   0.26   0.14
# 2     a         TRUE   0.27   0.77
# 3     a        FALSE   0.18   0.34
# 4     a        FALSE   0.23   1.63
# 5     a        FALSE   1.47   0.59
# 6     b        FALSE   0.33   0.46
# 7     b         TRUE   0.52   0.90
# 8     b         TRUE   0.86   2.50
# 9     b        FALSE   0.53   0.97
# 10    b         TRUE   0.15   0.03
# ...
# ...


# Objective: find a modularized way to calculate the standard deviation
#            for all columns that contain concentration measurements
```

```r
# create empty vector
std_devs <- c()
# loop over all columns
for (c in names(nutrients))
{
  # if column name equals "conc_N" calculate sd
  if (c == "conc_N"){
    cmean <- mean(nutrients[[c]])
    dev <- sum((nutrients[[c]] - cmean) **2)
    n=length(nutrients[[c]])-1
    fraction <- dev/n
    sd <- sqrt(fraction)
    # add result to vector
    std_devs <- c(std_devs, sd)
  } else{}
  # if column name equals "conc_P" calculate sd
  if (c == "conc_P"){
    cmean <- mean(nutrients[[c]])
    dev <- sum((nutrients[[c]] - cmean) **2)
    n=length(nutrients[[c]])-1
    fraction <- dev/n
    sd <- sqrt(fraction)
    # add result to vector
    std_devs <- c(std_devs, sd)
  } else {
  }
}
```

```r
# create empty vector
std_devs <- c()
# loop over all columns
for (c in names(nutrients))
{
  # if column name equals "conc_N" calculate sd
  if (c == "conc_N"){
    cmean <- mean(nutrients[[c]])
    dev <- sum((nutrients[[c]] - cmean) **2)
    n=length(nutrients[[c]])-1
    fraction <- dev/n
    sd <- sqrt(fraction)
    # add result to vector
    std_devs <- c(std_devs, sd)
  } else{}
  # if column name equals "conc_P" calculate sd
  if (c == "conc_P"){
    cmean <- mean(nutrients[[c]])
    dev <- sum((nutrients[[c]] - cmean) **2)
    n=length(nutrients[[c]])-1
    fraction <- dev/n
    sd <- sqrt(fraction)
    # add result to vector
    std_devs <- c(std_devs, sd)
  } else {
  }
}
```

```r
standard_deviation <- function(column){
  # check if the column contains numeric values
  # otherwise we do not calculate a standard deviation
  if (!is.numeric(column)){
    return()
  }
  # calculate the numerator of the fraction
  sum_squared_dist <- sum((column - mean(column)) ** 2)
  # Using N - 1 instead of N to get the corrected sample variance
  sample_size <- length(column) - 1
  sd <- sqrt(sum_squared_dist / sample_size)
  return(sd)
}


sapply(nutrients, standard_deviation)

```

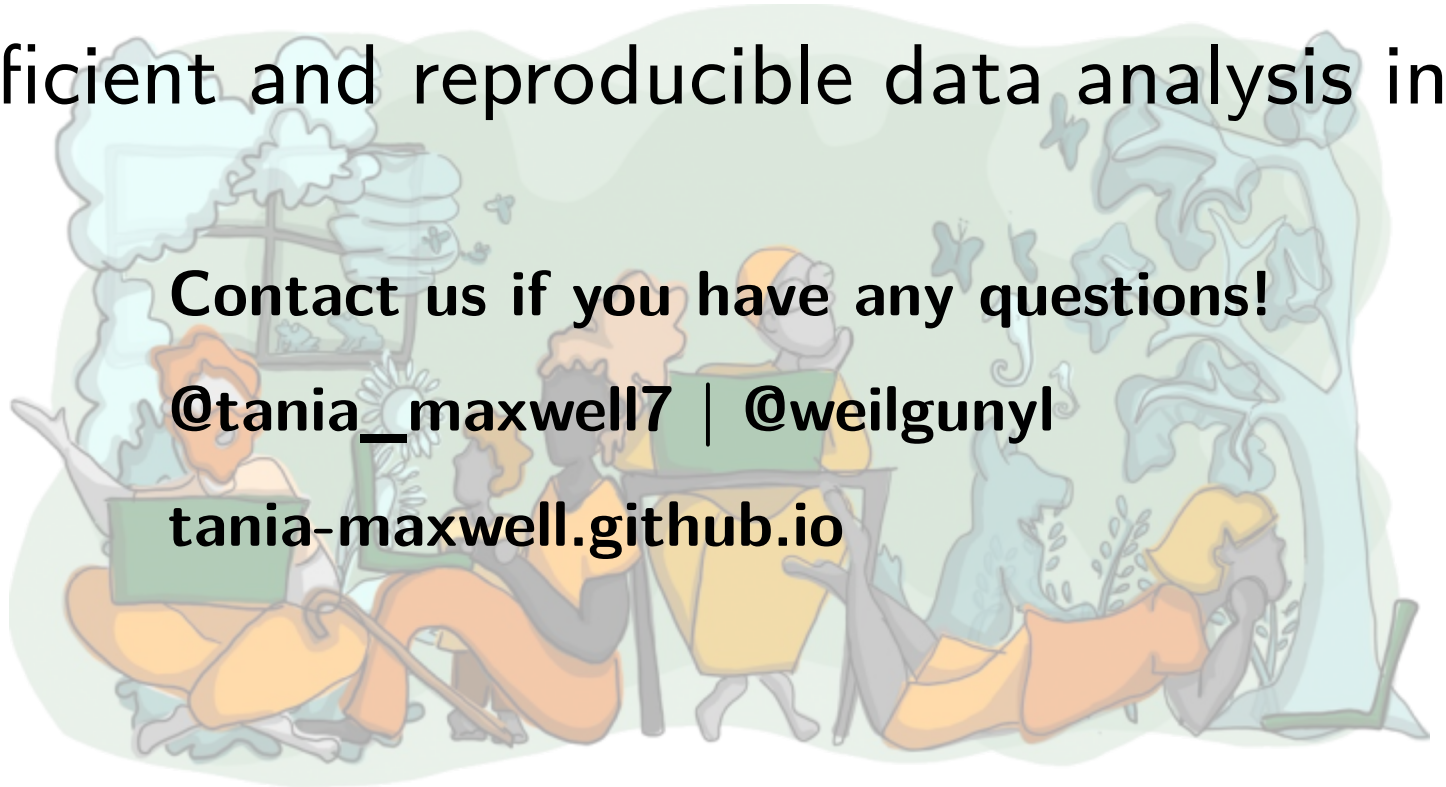
- Duplicate code → function
- Consistent style
- Useful comments & variable names
- General enough for re-use elsewhere

```r
standard_deviation <- function(column){
  # check if the column contains numeric values
  # otherwise we do not calculate a standard deviation
  if (!is.numeric(column)){
    return()
  }
  # calculate the numerator of the fraction
  sum_squared_dist <- sum((column - mean(column)) ** 2)
  # Using N - 1 instead of N to get the corrected sample variance
  sample_size <- length(column) - 1
  sd <- sqrt(sum_squared_dist / sample_size)
  return(sd)
}


sapply(nutrients, standard_deviation)

```

Efficient and reproducible data analysis in R



Contact us if you have any questions!

@tania_maxwell7 | @weilgunyl

tania-maxwell.github.io